INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Parallel bottleneck in the Quasineutrality solver embedded in GYSELA

G. Latu — N. Crouseilles — V. Grandgirard

N° 7595

Avril 2011

Domaine 1

ISSN 0249-6399 ISRN INRIA/RR--7595--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
NANCY - GRAND EST

Parallel bottleneck in the Quasineutrality solver embedded in GYSELA

G. Latu * †, N. Crouseilles †, V. Grandgirard *

Domaine : Mathématiques appliquées, calcul et simulation
Équipes-Projets CALVI

Rapport de recherche n° 7595 — Avril 2011 — 11 pages

Abstract: This report shows some performance results of the Quasineutraliy Poisson solver used in the GYSELA code. The numerical schemes of this Poisson solver is explained, and the computation and communication steps on a parallel machine are described. Benchmarks shows several time measurement from 32 cores to 4096 cores. Present bottlenecks and problems of the parallel algorithm are discussed. Some possible solutions are foreseen.

Key-words: Quasineutrality solver, Gyrokinetics, MPI

* CEA Cadarache, 13108 Saint-Paul-les-Durance Cedex
† INRIA Nancy-Grand Est & Université de Strasbourg, 7 rue Descartes, 6700 Strasbourg

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Limitations associées à la parallélisation du solveur Quasi-neutre inclus dans GYSELA

Résumé : Ce rapport présente des prises de performances du solveur Poisson Quasi-neutre utilisé dans le code GYSELA. Le schéma numérique de ce solveur Poisson est décrit, ainsi que les différentes étapes de calculs et de communication sur machine parallèle. Une série de benchmarks on été effectués de 32 à 4096 cœurs, cela donne un aperçu des performances de ce solveur parallèle. Les goulots d'étranglement et les limitations de l'algorithme parallèle utilisé sont explicités. Enfin, des solutions possibles sont envisagées.

Mots-clés : Solveur Quasi-neutre, Gyrocinétique, MPI

1 Introduction

Modeling turbulent transport is a major goal in order to predict confinement issues in a tokamak plasma such as ITER. Improving theoretical knowledge about the understanding of turbulent phenomena requires the description of particles considering their distribution in velocity. The gyrokinetic GYSELA code has been developed in order to model the physical features and instabilities (mainly ITG up to now) that appear at some specific time and space scales. This code uses the gyrokinetic framework that considers a computational domain in five dimensions: 3D in space, and 2D in velocity.

In the GYSELA code, a semi-Lagrangian solver is used to integrate Vlasov equation [5]. A hybrid MPI/OpenMP paradigm is used to benefit from a large number of processors while reducing communication costs [6]. A quasineutrality Poisson equation is considered to get electric potential from ion density, assuming an adiabatic electron response within cylindrical surfaces for the moment. The parallelization of Vlasov and Poisson solvers are tightly coupled because large amount of distributed data are exchanged between these two parallel components. Also, the parallel domain decomposition chosen for these solvers has an impact on performance. This document firstly describes the numerical scheme and implementation of the Poisson solver. Then, a brief overview of performances up to 4096 cores are shown. Finally, bottlenecks and possible solutions are discussed. This report follows a previous one on the same subject [7] where no OpenMP issues were mentioned.

2 Field solver of GYSELA

2.1 Quasineutrality equation

In tokamak configurations, the plasma quasineutrality approximation is currently assumed ([5, 8]). This leads to $n_i = n_e$ where n_i (resp. n_e) is the ionic (resp. electronic) density. On the one side, electron inertia is ignored, which means that an adiabatic response of electrons are supposed. On the other side, the ionic density splits into two parts. Using the notation $\nabla_\perp = (\partial_r, \frac{1}{r}\partial_\theta)$, the so-called linearized polarization density n_{pol} writes

$$n_{pol}(r,\theta,\varphi) = -\nabla_\perp . \left[\frac{n_0(r)}{B_0} \nabla_\perp \Phi(r,\theta,\varphi) \right],$$

where n_0 is the equilibrium density and B_0 the magnetic field at the magnetic axis. Second, the guiding-center density n_{Gi} is

$$n_{Gi}(r,\theta,\varphi) = 2\pi \int B(r,\theta) d\mu \int dv_{//} \mathcal{J}(k_\perp \sqrt{2\mu}) \bar{f}(r,\theta,\varphi,v_{//},\mu), \qquad (1)$$

where B is the magnetic field, $\bar{f}$ denotes the ionic distribution function evolving through a Vlasov type equation, $v_{//}$ the parallel velocity, μ the magnetic momentum and $\mathcal{J}$ is the gyroaverage operator.

Hence, the QN equation can be written in dimensionless variables

$$-\frac{1}{n_0(r)} \nabla_\perp . \left[\frac{n_0(r)}{B_0} \nabla_\perp \Phi(r,\theta,\varphi) \right] + \frac{1}{T_e(r)} \left[\Phi(r,\theta,\varphi) - \langle \Phi \rangle_{\theta,\varphi}(r) \right] = \tilde{\rho}(r,\theta,\varphi) \qquad (2)$$

with T_e the electronic temperature and where the definition of $\tilde{\rho}$ is given by

$$\tilde{\rho}(r,\theta,\varphi) = \frac{2\pi}{n_0(r)} \int B(r,\theta) d\mu \int dv_{//} \mathcal{J}(k_\perp \sqrt{2\mu})(\bar{f} - \bar{f}_{eq})(r,\theta,\varphi,v_{//},\mu). \quad (3)$$

In this last equation, $\bar{f}_{eq}$ denotes an ionic local Maxwellian equilibrium, and $\langle\rangle_{\theta,\varphi}$ the average onto the variables θ, φ.

The field solver includes several parts:

- First, the function $\tilde{\rho}$ is derived taking as input the function $\bar{f}$. Specific methods, not described here, are used to evaluate the gyroaverage operator $\mathcal{J}$ acting on $(\bar{f} - \bar{f_{eq}})$ in Eq. (3).
- Second, the 3D potential Φ is found in computing discrete fourier transforms of $\tilde{\rho}$, followed by the solving of tridiagonal systems and inverse fourier transforms.
- Finally, because of parallel work and data distributions during the previous steps, subdomains of Φ have to be exchanged between processors.

In the sequel, we will discuss about the parallelization of these parts. In the GYSELA code, this field solver is followed by another important computation step, that produces derivatives of Φ along each spatial direction. The computation of these derivatives are needed in order to set up processes for the Vlasov solver that comes next.

2.2 1D Fourier transforms method

The main benefit of the method described hereafter (from a parallel work distribution point of view) is to consider only 1D FFTs in θ dimension instead of 2D FFT in (θ, φ), and to uncouple hardly all computations in φ direction. The first description of this numerical scheme appears in [7].

The equation (2) averaged on (θ, φ) gives :

$$-\frac{\partial^2 \langle\Phi\rangle_{\theta,\varphi}(r)}{\partial r^2} - [\frac{1}{r} + \frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial \langle\Phi\rangle_{\theta,\varphi}(r)}{\partial r} = \langle\tilde{\rho}\rangle_{\theta,\varphi}(r) \quad (4)$$

A fourier transform in θ direction gives:

$$\begin{array}{ll} \Phi(r,\theta,\varphi) = & \sum_u \hat{\Phi}^u(r,\varphi) e^{i\,u\,\theta} \\ \tilde{\rho}(r,\theta,\varphi) = & \sum_u \hat{\rho}^u(r,\varphi) e^{i\,u\,\theta} \end{array} \quad (5)$$

The equation (2) could be rewritten as:

for $u > 0$:

$$-\frac{\partial^2 \hat{\Phi}^u(r,\varphi)}{\partial r^2} - [\frac{1}{r} + \frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial \hat{\Phi}^u(r,\varphi)}{\partial r} + \frac{u^2}{r^2}\hat{\Phi}^u(r,\varphi) + \frac{\hat{\Phi}^u(r,\varphi)}{Z_i\,T_e(r)} = \hat{\rho}^u(r,\varphi) \quad (6)$$

for $u = 0$:

$$\frac{\partial^2 \langle\Phi\rangle_\theta(r,\varphi)}{\partial r^2} - [\frac{1}{r} + \frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial \langle\Phi\rangle_\theta(r,\varphi)}{\partial r} + \frac{\langle\Phi\rangle_\theta(r,\varphi) - \langle\Phi\rangle_{\theta,\varphi}(r)}{Z_i\,T_e(r)} = \langle\tilde{\rho}\rangle_\theta(r,\varphi) \quad (7)$$

The equation (4) allows one to directly find out the value of $\langle\Phi\rangle_{\theta,\varphi}(r)$ from the data $\langle\tilde{\rho}\rangle_{\theta,\varphi}(r)$. Let us define the function $\Upsilon(r,\theta,\varphi)$ as $\Phi(r,\theta,\varphi) - \langle\Phi\rangle_{\theta,\varphi}(r)$.

Substracting equation (4) to equation (7) leads to

$$-\frac{\partial^2 \langle \Upsilon \rangle_\theta (r,\varphi)}{\partial r^2} - [\frac{1}{r} + \frac{1}{n_0(r)} \frac{\partial n_0(r)}{\partial r}] \frac{\partial \langle \Upsilon \rangle_\theta (r,\varphi)}{\partial r} + \frac{\langle \Upsilon \rangle_\theta (r,\varphi)}{Z_i\, T_e(r)} = \langle \tilde{\rho} \rangle_\theta (r,\varphi) - \langle \rho \rangle_{\theta,\varphi} (r) \quad (8)$$

Let us notice that $\hat{\Phi}^0(r,\varphi) = \langle \Upsilon \rangle_\theta(r,\varphi) + \langle \Phi \rangle_{\theta,\varphi}(r)$. So, the solving of equations (4) and (8) allows one to compute $\langle \Phi \rangle_{\theta,\varphi}(r)$, $\langle \Upsilon \rangle_\theta(r,\varphi)$ and $\hat{\Phi}^0(r,\varphi)$ from the quantities $\langle \tilde{\rho} \rangle_\theta(r,\varphi)$ and $\langle \tilde{\rho} \rangle_{\theta,\varphi}(r)$.

On the other hand, the equation (6) is sufficient to compute $\hat{\Phi}^{u>0}(r,\varphi)$ from $\tilde{\rho}$. So one can very early start the computations associated with $\hat{\Phi}^{u>0}(r,\varphi)$. The different equations are solved using a LU decomposition. Moreover, variable φ acts as a parameter in equation (6), allowing computations to be parallelized (each core can manage computations associated to one φ value).

2.3 Parallel algorithm using 1D FFT

The following algorithm is the most parallel, up to now, implemented in GYSELA[1]. We will work with the intermediate function $\Upsilon(r,\theta,\varphi) = \Phi(r,\theta,\varphi) - \langle \Phi \rangle_\theta (r,\varphi)$. The main idea is to get $\langle \Phi \rangle_{\theta,\varphi}$ for solving eq. (8) and then to uncouple computations along φ direction in the Poisson solver (Computation task 3 in the following list). The computation sequence is:

- Computation task 1
  - integrate $\bar{f}$ over $v_\parallel$ direction
- Computation task 2
  - compute $\tilde{\rho}(r=*,\theta=*,\varphi=*)$ in summing over μ direction
- Computation task 3
  - perform averages $\langle \tilde{\rho} \rangle_\theta(r=*,\varphi=*)$ and $\langle \tilde{\rho} \rangle_{\theta,\varphi}(r=*)$
  - get $\langle \Phi \rangle_{\theta,\varphi}$ from $\langle \tilde{\rho} \rangle_{\theta,\varphi}$ thanks to eq. (4)
  - For each φ value
    - ◇ FFT in θ on $\tilde{\rho}$
    - ◇ derive $\hat{\Phi}^u$ modes ($\forall u > 0$) with eq. (6)
    - ◇ compute $\langle \Upsilon \rangle_\theta(r=*,\varphi)$ with eq. (8)
    - ◇ add $\langle \Phi \rangle_{\theta,\varphi} + \langle \Upsilon \rangle_\theta(r=*,\varphi)$ to get $\hat{\Phi}^0(r=*,\varphi)$
    - ◇ inverse FFT in θ on $\hat{\Phi}$

[1] However, this formulation is not valid in the $b^\star$ version of GYSELA and is only used for **slab geometry** runs. Let us notice also, that the description of the $b^\star$ version of Poisson solver is not yet published

Algorithm 1: Full Parallelization of QN solver (`f1d_fft`)

1 *Input :* local block $\bar{f}(r = block, \theta = block, \varphi = *, v_{//} = *, \mu)$
2
3 *(* task 1*)*
4 *Computation :* $\tilde{\rho}_1$ by integration in $dv_{//}$ of $\bar{f}$
5 (parallelization in μ, r, θ)
6 **Send** local data $\tilde{\rho}_1(r = block, \theta = block, \varphi = *, \mu)$
7 **Redistribute** $\tilde{\rho}_1$ / **Synchronization**
8 **Receive** block $\tilde{\rho}_1(r = *, \theta = *, \varphi = block, \mu = *)$
9
10 *(* task 2*)*
11 **for** *local* φ *values* **do in parallel**
12 (parallelization in φ)
13 *Computation :* from $\tilde{\rho}_1$ for a given φ, computes $\tilde{\rho}_2$ by applying $\mathcal{J}$
14 (Fourier transform in θ, Solving of LU systems in r)
15 *Computation :* $\tilde{\rho}$ for a given φ by integration in $d\mu$ of $\tilde{\rho}_2$
16 *Computation :* accumulation of $\tilde{\rho}$ values to get $\langle\tilde{\rho}\rangle_\theta(r = *, \varphi)$
17 **end**
18 **Send** local data $\langle\tilde{\rho}\rangle_{\theta(r=*,\varphi=block)}$
19 **Broadcast of** $\langle\tilde{\rho}\rangle_\theta$ / **Synchronization**
20 **Receive** $\langle\tilde{\rho}\rangle_{\theta(r=*,\varphi=*)}$
21
22 *(* task 3*)*
23 *Computation :* Solving of LU system to find $\langle\Phi\rangle_{\theta,\varphi}$ from $\langle\tilde{\rho}\rangle_\theta$, eq. (4)
24 **for** *local* φ *values* **do in parallel**
25 (parallelization in φ)
26 *Computation :* 1D FFTs of $\tilde{\rho}$ on dimension (θ)
27 *Computation :* Solving of LU systems for $\hat{\Phi}$ modes ($u > 0$), eq. (6)
28 *Computation :* Solving of LU system for $\langle\Upsilon\rangle_{\theta(r=*,\varphi)}$, eq. (8)
29 *Computation :* Adding $\langle\Phi\rangle_{\theta,\varphi}$ to $\langle\Upsilon\rangle_{\theta(r=*,\varphi)}$ gives $\hat{\Phi}^0{}_{(r=*,\varphi)}$
30 *Computation :* inverse 1D FFTs on $\hat{\Phi}^0$ and $\hat{\Phi}^{u>0}$ to get $\Phi_{(r=*,\theta=*,\varphi)}$
31 **end**
32 **Send** local data $\Phi(r = *, \theta = *, \varphi = block)$
33 **Broadcast of values** / **Synchronization**
34 **Receive** global data $\Phi(r = *, \theta = *, \varphi = *)$
35 *Outputs :* $\Phi(r = *, \theta = *, \varphi = *)$

Let P be the number of cores used for a simulation run. From line 6 to line 8, a 4D function $\tilde{\rho}_1$ is redistributed. The amount of communication represents exchange of $N_r N_\theta N_\varphi N_\mu$ floats. The broadcast of lines 18 to 20 corresponds to a small communication cost of $N_r N_\varphi \max(P, N_\varphi)$. The final communication of lines 30-32 can be the biggest one (if P value is large enough) and induces the sending of $N_r N_\theta N_\varphi P$ floats.

3 Performance Analysis

3.1 Coarse view

Performances of the different versions of the QN solver for one 5D test case are presented in Table 1 ($N_r = 256, N_\theta = 64, N_\varphi = 256, N_{v_\parallel} = 16, N_\mu = 32$). Note that the Vlasov solver of the gysela code uses a parallelization based on a domain decomposition in dimensions μ and r. So, the number of cores P is given by the product of p_μ the number of μ values with p_r the number of block in dimension r. The number of μ values in the presented test case is $p_\mu = 32$, then our parallel program uses a minimum of 32 cores. Then, the relative speedups shown in Table 1 considers as a reference the execution times on 32 cores of four computation nodes. For this runs, **no OpenMP** parallelization were used, in order to show better the performance of the MPI parallelization only.

In order to give understandable performance results, we will subdivise the algorithms into distinguishable steps. The computation of $\tilde{\rho}$ (task 1 and task 2) can be decomposed in a communication part and many parallelized integral calculations. The Poisson solver giving Φ depending on $\tilde{\rho}$ subdivides into several communication steps, plus two types of computation: the redondant ones and the parallel ones. Finally, one can gather time costs of the QN solver in Table 1 considering three main mesures:

1. the largest time spent in communication among all cores,
2. the time spent in redondant-sequential computations,
   (each core has exactly the same work to do)
3. the time spent in parallel tasks.
   (the largest time spent among all cores is taken)

Numerical experiments of Table 1 were performed on a cluster of 932 nodes owned by CCRT/CEA, France. Each node hosts eight Itanium2 1.6Ghz cores and offers 24GB of shared memory.

Nb. cores.	32	128	256
p_r	1	4	8
communications	0.377 s	0.593 s	0.668 s
solve_seq	0.003 s	0.006 s	0.018 s
solve_par	4.078 s	1.039 s	0.528 s
Rel. speedup solve_par+_seq	1.0	3.9	7.7
Total	4.375 s	1.603 s	1.178 s
Rel. speedup	1.0	2.7	3.7

Table 1: Time measurements for one call to the QN solver in seconds and relative speedup are given (compared to performance of 32 cores with $p_r = 1$ and $p_\mu = 32$).

In this algorithm, almost all computations are parallelized and remaining sequential parts are negligible. The speedup of the computation part (`solve_par+_seq`) is impressive: 7.7 instead of 8 in the ideal case. The remaining parallel overhead comes from the small sequential computation of `solve_seq`

and above all communication `comm`. This QN solver is quite efficient and scalable in term of load balance, nevertheless communication costs are painful.

One could think about using other methods such as multigrid or a direct solver for the task 3 of the solver. But it won't diminish the cost of communications required for the calculation of $\tilde{\rho}$ (task 1), and for the final broadcast (task 3). So, we could not expect a large enhancement of parallel performance in such a way. One need firstly to try reducing the communication costs.

3.2 Detailed profiling

In order to interpret finely performance results, some detailed profiling is shown in this section. The test case considered here is bigger than in the previous subsection in order to be in a more realistic configuration. The GYSELA runs have used the following domain size:

$$N_r = 512, N_\theta = 256, N_\varphi = 128, N_{v_\parallel} = 32, N_\mu = 32 \ .$$

The **OpenMP** paradigm is now activated (which was not the case in the previous section) in order to further reduce computation times. All computations costs are lowered thanks to this fine grain parallelization. The main idea for this OpenMP parallelization has been to target φ loops. This approach is efficient for the computation task 1 (integrals in $v_\parallel$). But for computations task 2 and 3, this strategy competes with the MPI parallelization that uses also φ domain decomposition. Thus, above N_φ cores, no parallelization gain is expected. This fact is not a so hard constraint up to now, because communication costs are a lot more costly than computation tasks 2 and 3 (see Table 2).

Timing measurements have been performed on Jade2 machine (CINES) hosting 1300 Intel-X5560 nodes of eight cores. The Table 2 reports timing extracted from GYSELA runs. The smallest test case used 32 nodes (256 cores) while the biggest one has run on 512 nodes (4096 cores).

Nb. cores.	256	2048	4096
p_r	1	8	16
Algorithm			
comp1	8.9 s	0.80 s	0.41 s
io1	3.2 s	0.67 s	0.18 s
comp2	1.1 s	0.31 s	0.31 s
io2	0.50 s	0.49 s	0.02 s
comp3	0.04 s	0.05 s	0.03 s
io3	0.36 s	1.5 s	2.5 s

Table 2: Time measurements for one call to the QN solver in seconds

In Table 2, the `io1`, `io2`, `io3` steps states for communications associated with task 1, task 2, task 3 respectively. The communication costs for exchanging $\tilde{\rho}_1$ values (task 1) is reduced along with the involved number of nodes. This is explained by the fact that the overall network bandwidth is increased with large number of nodes, while the total amount of communications remains the same. The communication cost associated with `io2` is small and mainly consits of nodes synchronization. The `io3` communication involves a broadcast of electric

potential Φ on all nodes. On 4096 cores, the communication amount is bursting and represents a very large percentage of the timing cost of the QN solver.

Concerning computation costs, `comp1`, `comp2`, `comp3` stands for computation relative to task 1, task 2, task 3 respectively. The `comp1` calculation is the biggest CPU consumer of the QN solver; it scales well with the number of cores, combining MPI and OpenMP parallelizations. The `comp3` is the smallest computation step and time measures are nearly constant for all number of cores shown. In fact N_φ cores is the upper bound of the parallel decomposition of `comp3`, between 1 and N_φ cores the speedup increases, while the speedup and computation time are constant above this limit. The `comp2` step is quite in the same configuration than `comp3`. Nevertheless, cache effects explain why computation time is reduced going from 256 cores to 2048 cores.

Clearly, the main bottleneck of the QN solver is the `io3` communications, corresponding to the broadcast of electric potential Φ.

4 Perspectives

Wa can investigate some possible solutions to the bottleneck associated with communication costs. Several proposals are given in this section[2].

4.1 Suppress the final broadcast

The first idea that will certainly reduce the cost of `io3` is to send only the useful part of Φ to each node. The broadcast will be suppressed and replaced by selective send/receive to each nodes of Φ, and derivatives of Φ along spatial dimensions. The foreseen communications costs will be from one to three times of the observed `io1` communication cost.

4.2 Investigate other computation distributions

Let suppose we want to reduce the communication costs in improving locality of communications (on most architectures, nodes communicate faster with their direct neighbours). Then, we need surely to group cores that share the same spatial subdomain (r, θ, φ) but have different μ values, because of two reasons:

1. in task 1, one has to evaluate the integral in μ to compute $\tilde{\rho}_1$ on a given spatial subdomain (r, θ, φ);
2. in task 3, the final broadcast distributes the same set of data Φ to the cores owning different values of μ.

Without this new domain decomposition based firstly on spatial decomposition (higher levels of parallelism) and then to μ subdivision (lowest level of parallelism), it seems not possible to improve communication locality of $\tilde{\rho}_1$ computation.

Nevertheless this approach is very far away from present GYSELA version [6]. The parameter μ is placed at the highest level of parallelism, not at

[2] Acknowledgments: This work was partially suported by ANR EGYPT contract.

the lowest level one as proposed. The present μ parallelization improves the locality of communication in the Vlasov solver. So, if we change the parallel domain decomposition, it is likely that we will increase communication costs in the Vlasov part, while hoping a decrease of the communication cost in the QN solver.

4.3 Compression

One way to reduce the communication costs can be to compress the $\tilde{\rho}$ and the Φ data structures. A lossy or lossless compression on (r, θ) slices can be easily implemented while preserving the present parallelization scheme.

4.4 Redondant computations

If one considers a very large number of cores to run GYSELA, redondant computations could be foreseen in order to improve the final communication step. Suppose that we have at least $N_\varphi \times N_\mu$ cores (approximately 8192 cores for the biggest case ever run with GYSELA up to now). Then, each communicator responsible for one μ value can compute on N_φ cores the QN solver (computation task 3). Doing so, global computation costs are increased by a factor N_μ, but computation time remains identical to the present timings in computation task 3. Indeed, each core is responsible for computing at most one slice $\Phi(r = *, \theta = *, \varphi = value)$. The only difference is that N_μ cores will redondantly compute the same slice $\varphi = value$. By the way, the final communication involving all cores is avoided, because the potential Φ is known in each μ-communicator at the end of the computation task 3. Then, communications of `io3` step will take place more locally (only processors in the vincinity communicate) than at the present day, but with the same amount of overall communications.

References

[1] A.J. Brizard, T.S. Hahm, *Foundations of nonlineargyrokinetic theory*, PPPL report 4153, 2006.

[2] A.M. Dimits et al., *Comparisons and physics basis of tokamak transport models and turbulence simulations*, Phys. Plasmas **7**, pp. 969-983, (2000).

[3] G.W. Hammet, F.W. Perkins, *Fluid models for Landau damping with application to the ion-temperature-gradient instability*, Phys. Rev. Lett. **64**, pp. 3019-3022, (1990).

[4] S. Jolliet, A. Bottino, P. Angelino, R. Hatzky, T.M. Tran, B.F. Mcmillan, O. Sauter, K. Appert, Y. Idomura, L. Villard, *A global collisionless PIC code in magnetic coordinates*, Comp. Phys. Comm., **177**, pp. 409-425, (2007).

[5] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, Ph. Gendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrucker, J. Vaclavik, L. Villard, *A drift-kinetic Semi-Lagrangian 4D code for ion turbulence simulation*, J. Comput. Phys., **217**(2), pp. 395-423, (2006).

[6] G. Latu, N. Crouseilles, V. Grandgirard, E. Sonnendrücker, *Gyrokinetic semi-Lagarngian parallel simulation using a hybrid OpenMP/MPI programming*, Recent Advances in PVM an MPI, Springer, LNCS, pp. 356-364, Vol. 4757, (2007).

[7] G. Latu, V. Grandgirard, N. Crouseilles, R. Belaour, E. Sonnendrücker, *Some parallel algorithms for the Quasineutrality solver of GYSELA*, INRIA Research Report, RR-7591, (2011). *http://hal.inria.fr/inria-00583521/PDF/RR-7591.pdf*.

[8] T.S. Hahm, *Nonlinear gyrokinetic equations for tokamak microturbulence*, Phys. Fluids, **31**, p. 2670, 1988.

[9] R. Hatzky, T.M. Tran, A. Koenis, R. Kleiber, S.J. Allfrey, *Energy conservation in a nonlinear gyrokinetic particle-in-cell code for ion-temperature-gradient-driven modes in θ-pinch geometry* Physics of Plasma, Vol. 9, 2002.

[10] W.W. Lee, *Gyrokinetic approach in particle simulation*, Phys. Fluids **26**, p. 556, (1983).

[11] Z. Lin, W.W. Lee, *Method for solving the gyrokinetic Poisson equation in general geometry*, Phys. Rev. E **52**, p. 5646-5652, (1995).

[12] R.G. Littlejohn, J. Math. Phys. **23**, p. 742, (1982).

[13] Y. Nishimura, Z. Lin, J.L.V. Lewandowski, *A finite element Poisson solver for gyrokinetic particle simulations in a global field aligned mesh*, J. Comput. Phys, **214**, pp. 657-671, (2006).

[14] H. Qin, *A short introduction to general gyrokinetic theory*, PPPL report 4052, 2005.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399